Improving neural networks sturdiness through a data-augmentation generated by adversarial attacks

Guim González Torregrosa

Resumen– El propósito de este artículo es analizar qué son los ataques adversarios a las redes neuronales y si las redes pueden aprovechar estos ataques para obtener más datos de entrenamiento, aumentando la precisión y robustez de sus predicciones ante futuros ataques. También veremos si una entrada generada por un ataque a una red funciona en otras redes similares.
Para ello, se han generado más de 30 ataques adversarios y se probarán contra 3 arquitecturas de red diferentes. Una de las redes se volverá a entrenar con estos ataques y finalmente se probará para ver si estos ataques funcionan como aumento de datos y también si aumenta la robustez de la red.
Lo que veremos es que un ataque generado en una red A no funciona en otra red B. Este ataque confunde a la red B y la predicción obtenida es aleatoria pero no determinada. También veremos que los ataques adversarios se pueden utilizar como estrategias de aumento de datos y que las redes no se dejan engañar por el mismo tipo de ataques después de un paso de reentrenamiento. El último experimento nos mostrará que no hay un aumento de tiempo significativo en la ejecución de los ataques en una red reentrenada.

Palabras clave– redes neuronales, ataques adversarios, aumento de datos, deep learning, clasificación de imágenes

Abstract– The purpose of this article is to analyze what adversarial attacks on neural networks are and whether networks can take advantage of these attacks to obtain more training data, increasing their predictions accuracy and robustness to future attacks. We will also found if one input generated by an attack on one network works on other similar networks.
To do so, more than 30 adversarial attacks had been generated and will be tasted against 3 different network architectures. On of the networks will be re trained with this attacks and finally tested to see if this attacks work as data augmentation and also if the network sturdiness increases.
What we will see is that one attack generated on a network A does not work on some other network B. This attack confuses network B and the prediction obtained is random but not determined. We will also see that adversarial attacks can be used as data augmentation strategies and that the networks are not fooled by the same kind of attacks after a re-train step. The last experiment will show us that there is no significant time increase in the execution of the attacks on a re-trained network.

Keywords– neural networks, adversarial attack, data augmentation, deep learning, image classification

1 Introduction

Adversarial attacks show that many modern machine learning algorithms can be broken in surprising ways. These failures of machine learning demonstrate that even simple algorithms can behave very differently from what their designers intend. The objectives of the article are:

1. Understand what adversarial attacks are.
2. Apply the attack on a real world problem.
3. See if the input generated by an attack works in similar networks.
4. Use the generated attacks as data-augmentation for our model.

• Contact E-mail: guim.gonzalez@e-campus.uab.cat
• Menció realitzada: Computació
• Treball tutoritzat per: Ramón Baldrich (Computació)
• Curs 2020/21

5. Try to increase the sturdiness of our model by re-training it with the generated inputs.

6. Analyze the execution time before and after the re-train step of a given network.

Both point 1 and 2 are well known to the artificial intelligence community and there are numerous studies on them, which are referenced during the article.

The innovation of this research is found in the other ones. To see if an attack generated on one model can be used to attack other similar models, either because they have been trained with the same dataset or because they architecture is the same. And to see if the inputs generated with the attacks serve as data augmentation and later the execution time of the attack will increase if our model has become more robust.

2 State of the art

Currently there are many studies that are being carried out around adversarial attacks. Many of them intend to create tools or stickers that we can use in real life to be able to fool deep learning models.

One example would be glasses[7] with specific colors that when wearing them, a neural network could mistake our face for that of another person. This is a problem for security systems such as facial recognition since we can impersonate our victim.

Adversarial attacks are also applied to many fields such as gaming[3], by adding small perturbations to the input of the player agent, it can make the agent to miss-understand the context and do the exact opposite of what it is supposed to do. In the cited article there is an example of how an adversarial attack applied on Pong, makes the agent to miss the ball by going in the opposite direction.

It has also been proved the effectiveness of the attacks in printed format. The attack still works if you print the image generated, take a photo of the printing and classify it again with the same network. Imagine doing this technique with traffic signs (the dataset we use in this article), then printing the result and paste it in a real traffic sign. This will totally change the behaviour of self-driving cars.

There are studies in attack defenses[6] with different techniques such as modifying the data to simulate this perturbation in the original dataset, hiding the gradient and some other information about the model, or adding another class to the model as *NULL* and train the class with altered images.

This is not the case of this article. We do not intend to screw self-driving cars AI systems. Also we do not want to focus on new types of attacks. What we want to see is how we can take advantage of them and how they interact with other deep learning models.

3 Technologies used

The project is hosted in *Google Drive* and it makes use of the *Google Colab* computing power. Since all the project is based on deep learning models the need for a powerful GPU is a must. Although adversarial attacks are not high-cost algorithms in terms of computational resources, in some cases the attacks have collapsed the RAM of the computer and the GPU provided by *Google Colab*'s free plan. This is why it is recommended to replicate this project on a personal PC if you don't have a payment plan in cloud services.

The entire project has been developed with Python and the *Tensorflow* and *Keras* libraries for the creation of the models, the training process and the execution of the attacks. In the appendix section there is a link to the repository with the source code, the inputs generated and some analysis.

4 Adversarial Attacks

As described on the *OpenAI* blog[1]: *"Adversarial attacks are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake; they're like optical illusions for machines"*. This attacks are also hard to defend against because they require machine learning models to produce good outputs for *every possible input*. Most of the time, machine learning models work very well but only work on a very small amount of all the many possible inputs they might encounter.

In this section we will demonstrate what an adversarial attack is and how it is done. To do so, we will take the example from *Explaining and Harnessing Adversarial Examples*[2] as a reference.

In the article cited before, it uses an image of a panda along with a small perturbation. The result of this mix confuses the model and the output is something else completely different from what we would expect, a gibbon in this case. In order not to copy the example, we will trick the model with a lemon and we will fool the image recognition model *InceptionV3* from *Google*. *InceptionV3* is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the *ImageNet* dataset.

To prepare our input image to match the shape of the input layer of the network, we have to resize it, normalize it, and add a fourth dimension. The resize will be of 299 pixels height and 299 pixels width. The normalization will be in range of [-1,1]. The fourht dimension we need to add is the *batch size* of the inputs. Since we are dealing with just one image our *batch size* will be 1. With all this, we will have an input shape of $[1, 299, 299, 3]$.

Once the input image is prepared we can begin the attack algorithm. Here is the result of our generated attack, converting a giant panda into a lemon, imperceptible to the human eye.

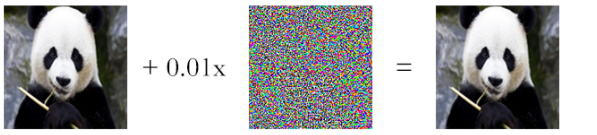

Fig. 1: Example of an adversarial attack generation applied to InceptionV3 on ImageNet. By adding a small perturbation equal to the gradient of the cost function with respect to the input, we can trick the model's classification of the image. In this examples and the ones ahead, the perturbation applied will be of 0.01 (all images are normalized between [-1,1] before entering the network).

Let x be our input image (the panda), p the maximum perturbation we accept in the input, we can define $pMax$ and $pMin$ being the maximum and minimum changes we can perform in the input image.

$$x = image \tag{1}$$
$$pMin = x - p \tag{2}$$
$$pMax = x + p \tag{3}$$

Since we want to convert our panda into a lemon, first we have to define our optimization process. We need to maximize the probability of obtaining the target class, so we will create a cost function equal to the last layer of the network with the target class as the result.

$$loss = output\_layer[0, target\_class] \tag{4}$$

This loss function will return us a vector of all the probabilities for each class. Each probability will be 0 except the target class position (the ideal result of the network for this class).

Once we have our loss function defined, we need to create our gradient. When we create a neural network, we want to find the gradient between the error and the parameters, this will tell us what values we need to change in our parameters to slowly minimize the error. But in this case we will use the gradient not on the parameters but on the input variable, the input layer (with the image tensor in it). Using the *Keras* function **gradients** we can obtain the graph of the gradients.

$$gradient, cost = K.gradients(loss, input\_layer) \tag{5}$$

In this $gradient$ variable we have a tensor that tells us which values we need to change in the input_layer tensor in order to optimize our loss function. In the $cost$ variable we have the cost associated to this gradient step. Adding the gradient to the image and clipping the result between the values $pMin$ and $pMax$ we can modify our input inside the boundaries defined and change the output a little bit.

$$x = x + gradient \tag{6}$$
$$x = clip(x, pMin, pMax) \tag{7}$$

This is just one iteration of the process. The changes produced by the gradient will be very small. What we have to do is to put this process inside a loop and make a stop condition based on the function cost. The cost will define how much confidence the network will have on your generated attack. In all examples in this article, the algorithm iterates until it reaches a cost of 0.95.

With the past attack we can observe several things:

- The attack does not need to know what architecture or what is the configurations of the weights of the network on which it is applied. Just by having access to the model and seeing what output generates for each input, the attack minimizes the perturbation in the input trying to maximize the changes in the output.
- By specifying the maximum cost of the attack algorithm, we can make the network more secure from our fake input than from the original one. In the example we see that the network ensures with 95.9% security that it is a panda, while after the attack it ensures with 99.8% security that it is a lemon.
- Knowing the classes that the model can recognize, we can generate new inputs that show the output that we want. To demonstrate this, below is a table with the same image of the panda applied to different attacks that generate different fake inputs chosen by the user.

TABLE 1: PREDICTIONS GENERATED BY INCEPTIONV3

Prediction	**Confidence**	**Attack time *(s)***
Giant panda	95.9%	-
Lemon	99.8%	*450*
Great white shark	99.9%	*420*
Llama	99.6%	*345*
Beer glass	99.9%	*255*

Examples of adversarial attacks generations applied to InceptionV3 on ImageNet. Different perturbations had been calculated and applied to the same input, obtaining different outputs of our choice. The only thing that changes is the targets vector provided to the algorithm in order to calculate the perturbation.

5 WORK SETUP

To add meaning to the article, instead of transforming animals into lemons, we will work with traffic signs. We will carry out adversary attacks focused on a real problem with self-driving cars. Imagine one of these attacks in real life, transforming a "Stop" sign into a "Speed limit (120km / h)" sign. Here we do have bigger consequences.

5.1 Dataset

For this scenario we will use the *German Traffic Sign Recognition Benchmark*[4] dataset. The GTSB is a single-image, multi-class classification problem with more than 50.000 images divided in 43 classes. Here is an example of the first 20 classes in the dataset.

Fig. 2: Example of all the 43 classes in the GTSB dataset. Classes are sorted left to right and top to bottom, so "Speed Limit (20km/h)" is class 0, "Speed Limit (30km/h)" is class 1, and so on so forth.

One first challenge encountered with the dataset is that the dataset itself is not symmetric. For example, for class 0 we have 210 images and for class 2 we have 2.250. Therefore we can not train our model with all the images, since class 2 will have more influence in the weights configuration of the network than class 0. Here is a histogram of the number of images per class.

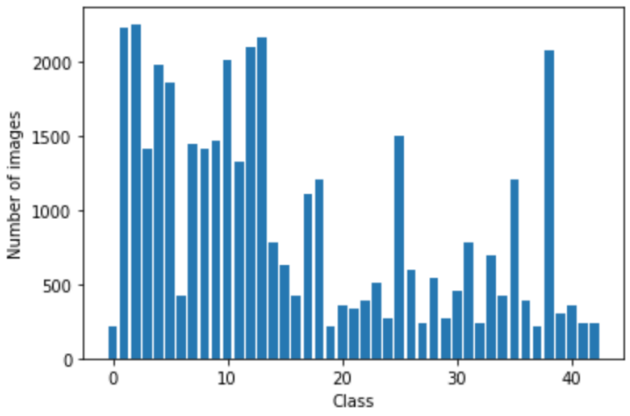

Fig. 3: Histogram showing the number of images for each class. The class with more images is number 2 with a total of 2.250 and the class with less images is number 0 with a total of 210.

To solve this, the models created during the project have been trained with 30 images of each class. They may seem few but as you will see in the training results we had enough with 30 images per class. This images have been selected randomly so as not to influence training.

5.2 Models

As explained at the beginning of the article, one of the objectives of the investigation is to see if an attack generated in one model has the same effect applied to a different model. To do this, we need to create and train different models with the use of *Tensorflow* and *Keras*.

All the models are Convolutional Neural Networks[5] with different architectures. Model number 1 will be used to generate all the attacks and will also be used for the retrain step (experiments 2 and 3). Models number 2 and 3 will be used to see if the attacks generated with number 1 also work on similar models (experiment 1).

The 3 networks are have been trained with the same dataset, the same images, badge size, test and train partitions, etc. The only difference are changes in the architecture of each one. All the code for the models is available at: $https://github.com/GuimGonzalez/adversarial-attacks-data-agumentation$.

Network architectures

Let C be a *2D Convolutional layer* with *ReLU* activation and k filters, F be a *Flatten layer* and D a *Dense layer* with size j, the architectures of the three neural networks are:

Network 1: $C64 - C64 - F - D256 - D43$
Network 2: $C64 - C32 - C16 - F - D256 - D43$
Network 3: $C32 - C16 - C64 - F - D256 - D43$

All the networks have a dropout of 0.5 between the two dense layers and they all use the *Adam* optimizer and the *categorical crossentropy* loss function.

The parameter used to save the model is the loss, every time the loss of one epoch is smaller than the previous one, a copy of the model is saved in *Google Drive*. As it is shown in the following chart, all the networks have acquired an accuracy of 99% and a loss of 0.01 (the charts are almost identical in the 3 networks so only *network 1* training history is shown).

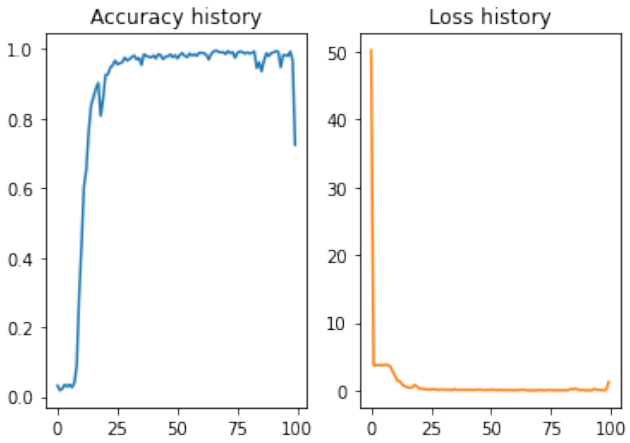

Fig. 4: Training history of network 1, showing charts of accuracy and loss during the training process. Note that the accuracy drop at the end is not significant since the model is only saved if some improvement is made in the loss.

The 3 networks have been trained with 30 images and the selection of them has been done randomly. The high accuracy achieved with only 30 images is due to the resizing to higher image resolution, the "easy" classification of traffic signs and the simplicity of the network.

5.3 Generation of attacks

Following the same steps explained before in the adversarial attack example of the panda, now we will do the same with our dataset. Here is an example of how a "Speed Limit (30km/h)" sign is converted into a "Speed Limit

(120km/h)". This is a big thread in real life, imagine being at the inside of your Tesla in auto pilot mode.

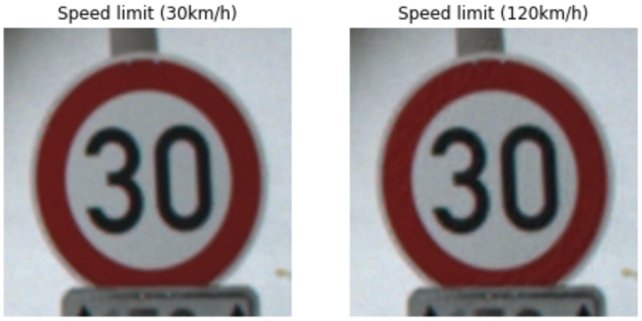

Fig. 5: Comparison of the original input and the input generated with an adversarial attack applied to our network. The human eye cannot see the differences between the two images since there is a maximum perturbation of 3 values per pixel.

Most of the images in the *GTSB* dataset have a size of 30×30 pixels (height per width). This means that every image has $30 \times 30 \times 3$ parameters, which is equal to 2700. With this amount of parameters the algorithm does not have enough freedom and most of the times it ends in an infinite loop. To solve this, there is a resizing process after every image is loaded, converting our initial 30×30 image into a 250×250 with 187500 parameters.

With this amount of parameters to play with, we see that even with a Grad-Cam[8] visual explanation of the attack there are not visual differences of what parts of the image are more important to the model in order to classify it correctly.

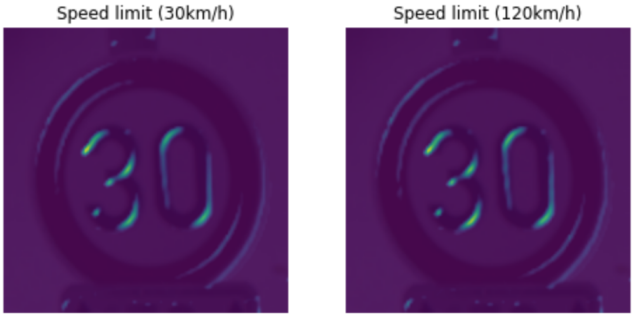

Fig. 6: Class activation visualization of the original image and the attack image using the Grad-Cam algorithm.

6 Experiment 1: Reliability

This experiment will test the reliability of the attacks. We want to see if an attack generated on a network A has the same impact on a network B. To do this, we have generated 34 attacks on *network 1*. If we test this attacks on *network 2* and *3*, we have a total of 68 tests, enough to see what happens with this first experiment. We can expect 3 different scenarios:

1. The attack failed, the network predicted the sign correctly.
2. The attack worked, the prediction is the same as the attack.
3. Network confused, the attack did not work but the prediction is something random.

After testing the 68 possibilities, we have obtained the following results:

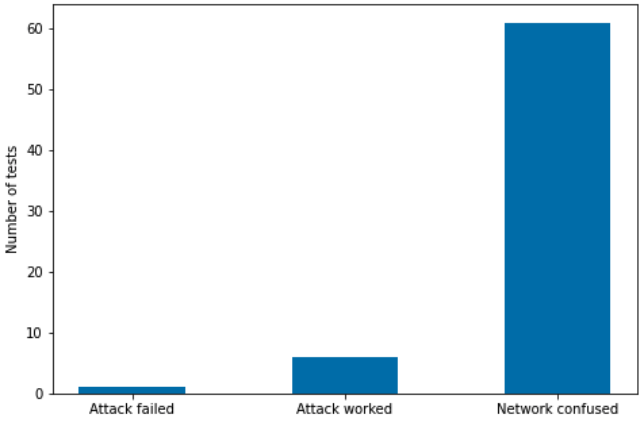

Fig. 7: Histogram showing how many attacks applied to other deep learning models had worked, confused the network or did not work.

What this histogram is showing us is that in fact, an adversarial attack generated on one network still confuses similar networks. But, if we want to get a specific result, we can not rely on the attack. If we do not mind what the result is, we are mostly sure the attack will change the output of the network.

7 Experiment 2: Data augmentation and sturdiness

This experiment will demonstrate if we can use generated attacks as new data for our model in subsequent training cycles. As we do not want to influence the training in any way, we will use attacks generated on the same type of signal, only one. If we used all the attacks that we have generated with different signals, we could make the network more robust with that type of signal than another.

Since we have 25 different generated attacks on "Speed limit (30km/h)" signal, we will use this images for our experiment. The first step is to split this images into train and test. We will use 20 images for the training step and 5 images for the test step. The images will be picked randomly to not influence the process. Once we have this two sets, it is time to train the network (we use network 1 generated previously) with the new images. Here is accuracy and loss history of the training phase.

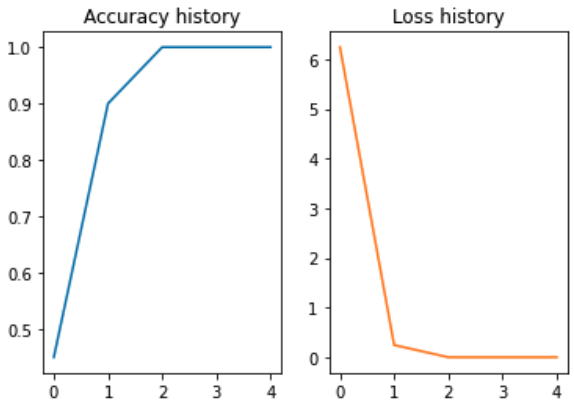

Fig. 8: Accuracy and Loss history of the retrain step, using generated attacks as inputs and specifying the desired output that we expect.

It is normal to see the accuracy starting at 0 and the loss starting at 6 even if the network was previously trained. This doesn't mean it is not working, in fact, this is what we should expect. Remember our goal with the attacks was to fool the machine, and to change the input image until we did not get our desired output. Since we are telling the network that this is now a completely different output, the accuracy falls to 0. It is like a child when learning something new from scratch.

Now that the network has been re-trained, we can test the rest of the attacks and see if the model is more robust or not. We can expect 3 different scenarios:

1. The attack worked, the prediction is the same output the attack generated.

2. Network confused, the attack did not work but the prediction is something random.

3. The attack failed, the network predicted the "Speed limit (30km/h)" signal.

After classifying the other images with the re-trained model, the rest of the attacks had totally failed. The network has learned with the previous attacks and now it is capable of predicting the "Speed limit (30km/h)" signal again without being fooled. Due to the lack of time, the number of tests are not as high as I would like, and therefore it requires more investigation on it. But for now, we can say two things:

- The network sturdiness has augmented. The attacks that once fooled the model, can not anymore. This means that we can make neural networks like the ones from Tesla self-driving models to increase their sturdiness against real attacks.

- Since they don't confuse the network anymore, this is an alternative way to generate a data augmentation set of images. We can create more images based on generated attacks, and another good thing is that we can generate as many as we want (with different perturbations). This gives us more possibilities when working with small datasets.

This are the parameters used for the retraining step:

- Epochs: 5

- Batch size: 10

- Validation data: same as training data

In this step we used the training data as validation data too. This is a bad practice in most cases since we overfit the network. But in this case it is exactly what we are trying to do, our need is to "fix" the network and change this predictions even using brute force.

8 Experiment 3: Attack execution time

With this last experiment we want to see if the execution time of an adversarial attack increases after re-training our model with these adversarial attacks. The steps we will follow to do this are these:

1. Take all the images used to re-train the model, 20 in total. This are the ones we want want to repeat since the network the weights of the network have been reconfigured with this images as the input.

2. Take the original images of the attacks we have selected. Remember we want to repeat the attack with the same conditions, except for the re-trained model. This means our input will be the original image, not the one generated from the attack. Since we have generated 3 attacks for every original image, we have 7 images to do the attack.

3. Repeat the attack on the re-trained model with this 7 images and compare the execution times with the old ones. For every image we can generate 3 attacks, the same ones as before.

After doing all this steps and repeating the 20 attacks, 60% of them have increased the execution time and 40% of them have reduced the execution time.

Taking into account the small number of images with which the experiment has been carried out, we can say that after the re-train step even if the network sturdiness has improved and it is not fooled anymore by the same attacks, the execution time still remains random. There is no enough evidence to say that the execution time has increased since we do not have a significant differences in the repeated attack times.

This result was to be expected since, as we have seen before with the Grad-Cam, the network does not look at a different part of the image. This perturbation is so small and so random that it is not about a fixed part of the image but some exact values which need to be found.

In fact, if we analyze an attack history chart, we can see that the cost does not increase until the algorithm finds which pixels have to be altered.

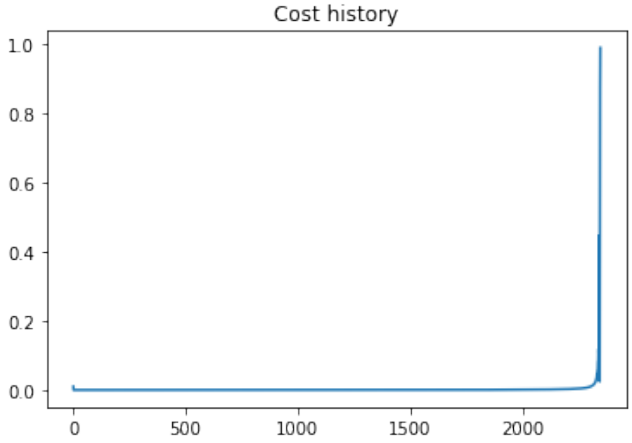

Fig. 9: History showing the cost of an adversarial attack. It shows how most of the time spent in an adversarial attack is in finding the exact pixels it has to alter in order to get the desired output.

Most of the time of an adversarial attack algorithm is spent in finding this values. Once this threshold is passed and the algorithm fins them and sees that the gradient starts to change with a higher frequency, it only takes a few steps to generate the result image.

9 Conclusions

The general purpose of this article was to analyze adversarial attacks and see what benefits we can obtain from them, such as data-augmentation or an increase in the robustness of the network. On top of that, we have been able to obtain observations from several different experiments. We have seen that:

- Adversarial attacks do not need to know what architecture of the network on which they are being executed. They treat the network as a black-box only knowing the input shape and the desired output. This is a threat to all available models in the Internet.
- By specifying the maximum cost of the attack algorithm, we can make the network more secure from our fake input than from the original one. The cost of the attack is the confidence threshold of our output.
- The size of the input data affects directly to the attack execution time. If the input is small, the attack does not have enough parameters to alter inside the perturbation defined and the execution never ends. Otherwise, if the input is so big, the attack spends so much time trying to find which are the correct values to change in order to get the desired output. In our case with the $250 \times 250 \times 3$ images we have gotten execution times from 0 seconds to 5 minutes.
- Adversarial attack generated on one network still confuses similar networks. If we want to get an specific output from a network, we have to apply the attack to that specific network. But if we just want to confuse the network, an adversarial attack generated on a similar network might confuse this second one. This results are not 100% exact so it is better to generate the attack always on the target network.
- It is possible to increase the network sturdiness to future attacks by re-training it with fake inputs. This leads us to new data-augmentation strategies. Since the network is not fooled anymore, we can use this inputs as new training data.
- Last, we have seen that most of the time of an adversarial attack algorithm is spent in finding which values have to be changed. This process with the gradient has a random component and so the attack execution time does not directly depend on the network sturdiness. This conclusion needs further investigation as it has not been done with the desired amount of data.

Adversarial attacks are hard to defend against because it is difficult to construct a theoretical model of the attack crafting process. Therefore, while there are many studies being carried in the defense against this kind of attacks, I encourage researchers to also study beneficial cases of having a way to generate new data for the machine, even if it looks the same to the human eye.

Acknowledgments

To my tutor Ramón Baldrich for the help provided during these months, the proposals for new hypotheses and the documentation provided.

To my family for encouraging me throughout the project.

References

[1] Ian Goodfellow, Nicolas Papernot, Sandy Huang, Rocky Duan, Pieter Abbeel, and Jack Clark. Attacking machine learning with adversarial examples, 2017.

[2] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[3] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents, 2019.

[4] Mykola. Gtsrb - german traffic sign recognition benchmark, 2018.

[5] Krupa Patel. Traffic sign recognition with keras and deep learning, 2020.

[6] Shilin Qiu, Qihe Liu, Shijie Zhou, and Chunjiang Wu. Review of artificial intelligence adversarial attack and defense technologies. *Applied Sciences*, 9(5), 2019.

[7] Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. Adversarial attacks and defenses in deep learning, 2020.

[8] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

APPENDIX

Adversarial attacks


Speed limit (30km/h)
Speed limit (100km/h)
Speed limit (30km/h)
End of speed limit (80km/h)
Speed limit (30km/h)
End of speed limit (80km/h)
No entry
End of speed limit (80km/h)
Speed limit (30km/h)
End of speed limit (80km/h)
No entry
Speed limit (100km/h)
Speed limit (30km/h)
Speed limit (100km/h)
No entry
Speed limit (120km/h)
Speed limit (30km/h)
Speed limit (120km/h)
Speed limit (30km/h)
End of speed limit (80km/h)
Speed limit (30km/h)
Speed limit (120km/h)
Speed limit (30km/h)
Speed limit (100km/h)


Speed limit (30km/h)
Speed limit (120km/h)
Speed limit (30km/h)
Speed limit (120km/h)
Speed limit (30km/h)
Speed limit (120km/h)
Speed limit (30km/h)
Speed limit (120km/h)
Speed limit (30km/h)
Speed limit (100km/h)
Speed limit (30km/h)
Speed limit (100km/h)
Speed limit (30km/h)
End of speed limit (80km/h)
Speed limit (30km/h)
End of speed limit (80km/h)
Speed limit (30km/h)
End of speed limit (80km/h)
Speed limit (30km/h)
Speed limit (100km/h)

Grad-Cam class activation visualization